■ อุรุพจน์ กัลยาสิริ

ต้านยูนิกซ์แครกเกอร์
: สร้างความปลอดภัยให้ FTP Server

การโอนย้ายแฟ้มข้อมูลเป็นบริการพื้นฐานที่สำคัญอย่างหนึ่งของอินเทอร์เน็ตเซิร์ฟเวอร์ ข้อมูลที่เปิดบริการให้คัดลอกจะเป็นแฟ้มข้อมูลที่ต้องการเผยแพร่ ทั้งนี้ก็เพื่อสร้างความสะดวกให้แก่ผู้ใช้งานอินเทอร์เน็ตสามารถถ่ายโอนแฟ้มได้ เซิร์ฟเวอร์จะอนุญาตให้ผู้ใช้ในเครือข่ายคัดลอกแฟ้มได้โดยสะดวกตราบเท่าที่ผู้นั้นสามารถติดต่อขอบริการผ่านทางเครือข่ายอินเทอร์เน็ตและได้รับอนุญาตจากเซิร์ฟเวอร์ที่ให้บริการ เราเรียกเซิร์ฟเวอร์ที่ให้บริการถ่ายโอนข้อมูลนี้ว่า **FTP Server** โปรโตคอลมาตรฐานที่สำคัญของชุดโปรโตคอล TCP/IP ชนิดหนึ่งคือ **FTP** : File Transfer Protocol ทำหน้าที่กำหนดรูปแบบการถ่ายโอนแฟ้มข้อมูลระหว่างผู้ใช้บริการและผู้ให้บริการ ช่วยให้การโอนย้ายแฟ้มข้อมูลไม่จำกัดอยู่กับประเภทของโฮสต์หรือระบบปฏิบัติการ นอกจากนั้นขั้นตอนวิธีตามมาตรฐานโปรโตคอล FTP ยังช่วยให้ข้อมูลที่ถ่ายโอนมีความถูกต้องและครบถ้วน

กล่าวถึงเฉพาะโปรแกรม FTP Serverที่นิยมใช้อย่างแพร่หลายคือ wu-ftp

การสร้างระบบรักษาความปลอดภัยของ FTP Server เป็นข้อควรคำนึงประการสำคัญของการให้บริการถ่ายโอนแฟ้มข้อมูล เนื่องจากการให้บริการนี้เปิดโอกาสให้ผู้ใดก็ได้ในเครือข่ายสามารถติดต่อขอใช้บริการ ดังนั้นจึงอาจจะมีผู้ที่ไม่ประสงค์ดีติดต่อขอเข้าใช้บริการแล้วอาศัยจุดอ่อนของระบบเข้ามาบุกรุกและสร้างความเสียหาย ดังนั้นผู้ดูแลระบบจึงควรสร้างระบบรักษาความปลอดภัยให้แก่ FTP Server วิธีการสร้างความปลอดภัยนั้นทำได้หลายแบบ ซึ่งแต่ละรูปแบบจะแตกต่างกันไปตามความสามารถของโปรแกรม FTP Server บางโปรแกรมก็มีจุดเด่นที่ดีและบางโปรแกรมก็มีจุดด้อยที่ต้องระวังเช่นกัน ในที่นี้จะขอกล่าวถึงเฉพาะโปรแกรม FTP Server ที่นิยมใช้อย่างแพร่หลายคือ **wu-ftp** ซึ่งเป็นโปรแกรมที่พัฒนาขึ้นใช้งานภายใต้ระบบปฏิบัติการลีนุกซ์ของชุดเผยแพร่เรดแฮต wu-ftp ทำงานอย่างมีประสิทธิภาพและมีฟังก์ชันให้ใช้มากมาย เช่น สามารถจัดแบ่งผู้ใช้

UNIX

เป็นคลาส เพื่อความสะดวกในการควบคุมและจัดการกับกลุ่มผู้ใช้ รวมทั้งมีคำสั่งให้ผู้ดูแลเซิร์ฟเวอร์กำหนดรูปแบบการทำงานเพื่อกำหนดนโยบายรักษาความปลอดภัยได้อย่างหลากหลาย

ติดตั้ง FTP Server

FTP Server ใช้ชื่อแดมอนว่า *ftpd* หรือ *in.ftpd* ในยูนิกซ์ทั่วไปแดมอนนี้อาจอยู่ที่ไดเรคทอรี */etc* หรือ */usr/etc* สำหรับในเรดแฮตจะเก็บอยู่ใน */usr/sbin* แดมอนมักถูกเรียกใช้ผ่าน *inetd* เมื่อตรวจสอบในแฟ้ม */etc/inetd.conf* จะพบบรรทัดคำสั่งดังนี้

```
# grep in.ftp /etc/inetd.conf
ftp  stream  tcp  nowait  root  /usr/sbin/tcpd  in.ftpd -l -a
```

FTP Server มีบริการให้กับผู้ใช้งานทั่วไปผ่านบัญชี *anonymous* ดังนั้นจึงจำเป็นต้องมีบัญชีประจำของเซิร์ฟเวอร์ในแฟ้ม */etc/passwd* เพื่อให้บริการดังกล่าว ชื่อบัญชีและรูปแบบปกติที่ใช้งานจะมีลักษณะดังนี้

```
# grep ftp /etc/passwd
ftp:x:14:50:FTP User:/home/ftp:null
```

จากข้อมูลข้างต้นจะพบว่าชื่อบัญชีคือ *ftp* โดยมีไดเรคทอรีอยู่ที่ */home/ftp* และเชลล์ประจำบัญชีคือ *null* ซึ่งเท่ากับว่าไม่อนุญาตให้มีการล็อกอินผ่านเทลเน็ต ทั้งนี้เพื่อช่วยรักษาความปลอดภัยระบบ ไดเรคทอรี */home/ftp* เป็นไดเรคทอรีระบบประจำ FTP Server ผู้ดูแลระบบจะบรรจุแฟ้มข้อมูลที่ต้องการให้ถ่ายโอนให้อยู่ภายใต้ไดเรคทอรีนี้

ต่อไปจะอธิบายถึงการปรับแต่งแฟ้มระบบ แต่เนื่องจากเรดแฮตได้จัดเตรียมแฟ้มระบบและค่าโดยปริยายไว้ให้แล้ว ผู้ดูแลเซิร์ฟเวอร์สามารถให้บริการ FTP ได้ทันทีโดยบรรจุแฟ้มที่ต้องการให้บริการในไดเรคทอรีย่อย *pub* เรดแฮตดำเนินการติดตั้ง FTP Server ให้โดยอัตโนมัติ เพื่อตรวจสอบการติดตั้งเซิร์ฟเวอร์ให้ดำเนินการตามขั้นตอนต่อไปนี้

1. ล็อกอินด้วย *root*

2. ตรวจแฟ้ม */etc/inetd.conf* ว่ามีการเรียกใช้แดมอนด้วยบรรทัดคำสั่งต่อไปนี้

```
ftp  stream  tcp  nowait  root  /usr/sbin/tcpd  in.ftpd -l -a
```

3. ออปชัน -l ในคำสั่งหมายถึงให้บันทึกการใช้งาน FTP ด้วย **syslog** ออปชัน -a คือให้ทำงานตามข้อกำหนดในแฟ้ม *ftpaccess*

4. ตรวจสอบสิทธิ์ประจำไดเรคทอรี *bin*, *etc*, *pub*, *lib* และ *incoming* เพื่อรักษาความปลอดภัยระบบ (ดูหัวข้อการกำหนดสิทธิ์การใช้ไดเรคทอรี)

5. สร้างแฟ้ม *.rhosts* และ *.forward* โดยไม่บรรจุข้อมูลใด และปรับสิทธิให้เป็น 400 เพื่อช่วยเรื่องความปลอดภัยระบบดังนี้

```
# cd ~ftp
# touch .rhosts .forward
# chmod 400 .rhosts .forward
```

6. ปรับแต่งและแก้ไขแฟ้ม *ftpconversions*, *ftphosts* และ *ftpaccess* เพื่อกำหนดรูปแบบการให้บริการ

7. บรรจุแฟ้มที่ต้องการให้บริการลงในไดเรคทอรี *pub*

ไดเรคทอรีรากประจำ FTP

เมื่อผู้ใช้ *anonymous* ขอเข้าใช้บริการ ไดเรคทอรีที่อนุญาตให้ใช้งานได้แก่ไดเรคทอรีบ้านของบัญชี *ftp* ตามที่กำหนดไว้ใน */etc/passwd* กรณีของเรดแฮตจะกำหนดให้เป็น */home/ftp* แต่เนื่องจาก *ftpd* จะเรียกใช้ฟังก์ชันระบบ **chroot()** ทำให้ผู้ใช้ที่เข้ามาใช้บริการเห็นเสมือนว่าทำงานอยู่ที่ไดเรคทอรีราก กล่าวอีกนัยหนึ่งคือ ไดเรคทอรี */home/ftp* จะถูกแม็พให้ผู้ใช้มองเสมือนเป็นไดเรคทอรีราก ภายใน /home/ftp นี้ประกอบด้วยไดเรคทอรีย่อยที่สำคัญ 3 ไดเรคทอรีคือ

bin เก็บโปรแกรมที่จำเป็นต้องใช้ในการให้บริการ ตัวอย่างเช่นคำสั่ง **ls**

etc ไดเรคทอรีเก็บรหัสผ่าน passwd (ต่างแฟ้มกับ /etc/passwd) เพื่อให้ ftpd ทราบถึงสิทธิของผู้เข้าใช้งาน นอกจากนี้ยังใช้ในการแสดงเจ้าของและกลุ่มที่แฟ้มข้อมูลสังกัดอยู่เมื่อผู้ใช้ขอดูแฟ้มด้วย **ls**

pub ไดเรคทอรีบรรจุแฟ้มที่ให้บริการถ่ายโอน

หมายเหตุ : เรดแฮตจะสร้างไดเรคทอรีภายใน */home/ftp* อีกสองไดเรคทอรี ไดเรคทอรีแรกคือ *lib* ใช้บรรจุโหลดเดอร์และคลังโปรแกรมภาษาซีเพื่อใช้ระหว่างการทำงาน ไดเรคทอรีที่สองคือ *incoming* สำหรับให้ผู้ใช้สามารถถ่ายแฟ้มมาไว้ใน FTP Server กรณีที่ไม่ต้องการให้บริการอัพโหลด ให้ลบไดเรคทอรี *incoming* ออกไปเพื่อรักษาความปลอดภัยให้แก่ระบบอีกทางหนึ่ง

การกำหนดสิทธิการใช้ไดเรคทอรี

เพื่อจำกัดสิทธิ์การใช้แฟ้มและระวังด้านความปลอดภัยของระบบ ผู้ดูแลเซิร์ฟเวอร์จำเป็นต้องตรวจและปรับแต่งสิทธิ์การใช้แฟ้มในไดเรคทอรีที่ให้บริการดังนี้

ไดเรคทอรีบ้าน เป็นเจ้าของโดย *root* และอยู่ในสังกัดกลุ่ม *wheel* หรือ *other* โดยเปิดสิทธิ์ให้อ่านและเปลี่ยนไดเรคทอรีได้สำหรับผู้ใช้ทุกคน (รหัส 555)

bin เป็นเจ้าของโดย *root* และอยู่ในสังกัดกลุ่ม *wheel* หรือ *other* โดยเปิดสิทธิ์ให้เรียกโปรแกรมได้อย่างเดียว ห้ามอ่านและเขียน (รหัส 111)

etc เป็นเจ้าของโดย *root* และอยู่ในสังกัดกลุ่ม *wheel* หรือ *other* โดยเปิดสิทธิ์ให้เรียกโปรแกรมได้อย่างเดียว ห้ามอ่านและเขียน (รหัส 111)

pub ควรเป็นเจ้าของโดยผู้ใช้ที่ได้รับมอบหมายจากผู้ดูแลระบบให้ดำเนินการบริหารข้อมูล ไดเรคทอรีนี้ต้องเปิดสิทธิ์ให้อ่านและเปลี่ยนไดเรคทอรีได้สำหรับผู้ใช้ทุกคน (รหัส 555)

incoming เป็นเจ้าของโดย root และอยู่ในสังกัดกลุ่ม wheel หรือ other โดยเปิดสิทธิ์ให้เขียน และเรียกโปรแกรมได้สำหรับผู้ใช้ทุกคน แต่ไม่ให้อ่าน ทั้งนี้เพื่อไม่ให้ผู้ใช้ทราบว่ามีแฟ้มใดถูกถ่ายโอนเข้ามาและป้องกันไม่ให้ลบแฟ้มนั้นได้ และยังป้องกันไม่ให้ใช้ FTP Server เป็นที่รับฝากข้อมูล คำสั่งที่ใช้ปรับสิทธิ์คือ **chmod 1733 incoming** หรือ **chmod +t incoming**

ไดเรคทอรีรากควรกำหนดสิทธิ์ตามตัวอย่างดังต่อไปนี้

```
# cd /home/ftp
# ls -l
total 5
d--x--x--x  2 root  wheel  1024 Aug 27  1997 bin
d--x--x--x  2 root  wheel  1024 Aug 27  1997 etc
drwx-wx-wt  2 root  wheel  1024 Sep 18  1998 incoming
drwxr-xr-x  2 root  wheel  1024 Aug 27  1997 lib
dr-xr-xr-x  3 root  wheel  1024 Apr 20 14:05 pub
```

ไดเรคทอรี *bin* และ *etc* อาจมีแฟ้มและกำหนดสิทธิ์ตามตัวอย่างดังต่อไปนี้

```
# ls -l bin
total 579
---x--x--x  1 root  wheel   14940  Aug 27  1997 compress
---x--x--x  1 root  wheel  292160  Aug 27  1997 cpio
---x--x--x  1 root  wheel   45056  Aug 27  1997 gzip
---x--x--x  1 root  wheel   49432  Aug 27  1997 ls
---x--x--x  1 root  wheel   56380  Aug 27  1997 sh
---x--x--x  1 root  wheel   77560  Aug 27  1997 tar
---x--x--x  1 root  wheel   45056  Aug 27  1997 zcat
# ls -l etc
total 6
-r--r--r--  1 root  wheel      64  Apr 21 09:04 group
-r--r--r--  1 root  wheel      79  Aug 27  1997 passwd
```

แฟ้มรหัสผ่านและกลุ่ม

ตามที่กล่าวมาแล้วว่าแฟ้ม *passwd* และ *group* ของ FTP ใช้ในการแสดงเจ้าของและกลุ่มที่แฟ้มข้อมูลสังกัดอยู่เมื่อผู้ใช้ขอดูแฟ้มด้วย **ls** กล่าวโดยละเอียดคือ FTP จะตรวจดูว่าหมายเลขรหัสของเจ้าของและกลุ่มแฟ้มนั้นมีค่าเท่าใด จากนั้นจะนำรหัสนั้นมาหาชื่อใหม่จากแฟ้ม *passwd* และ *group* เพื่อแสดงบนจอภาพ

ตัวอย่างต่อไปนี้เป็นข้อมูล *passwd* และ *group* ที่กำหนดมาโดยเรดแฮตและผ่านการแก้ไขโดยเพิ่มกลุ่ม other (กลุ่มหมายเลข 10)

/home/ftp/etc/passwd

```
root:*:0:0:::
bin:*:1:1:::
operator:*:11:0:::
ftp:*:14:50:::
nobody:*:99:99:::
```

/home/ftp/etc/group

```
root::0:
bin::1:
daemon::2:
sys::3:
adm::4:
other::10:
ftp::50:
```

เมื่อตรวจหมายเลขกลุ่ม *wheel* จาก */etc/passwd* จะพบว่ามีหมายเลขกลุ่มเท่ากับ 10 และเมื่อขอดูรายชื่อแฟ้มใน *pub* จะปรากฏสิทธิ์ดังนี้

```
# grep wheel /etc/group
wheel::10:root
# ls -l /home/ftp/pub
total 5                                  กลุ่ม wheel เมื่อดูผ่านเชลล์ปกติ
drwxr-xr-x   3 root     wheel        1024 Mar 23 09:23 archive
-rwxr--r--   1 root     wheel        1910 Apr 20 14:05 big
-rwxr--r--   1 root     wheel        1275 Apr 20 13:42 small.Z
```

แต่เมื่อดูผ่านบริการ FTP ด้วยคำสั่ง **dir** ปรากฏว่ากลุ่มที่สังกัดไม่ใช่ *wheel* แต่เป็น *other* ทั้งนี้เพราะ FTP ใช้รหัสหมายเลข 10 ซึ่งเป็นของ *wheel* แล้วเปิดหาชื่อจากตัวเลขนี้ในแฟ้ม *group* ซึ่งได้ชื่อ *other* จากนั้นจึงแปลงจาก *wheel* ไปเป็น *other* ดังตัวอย่างต่อไปนี้

```
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 7
dr-xr-xr-x   3 root     other        1024 Apr 20 07:05 .
drwxr-xr-x   8 root     other        1024 Apr 21 02:32 ..
dr-xr-xr-x   3 root     other        1024 Mar 23 02:23 archive
-rw-r--r--   1 root     other        1910 Apr 20 07:05 big
-rw-r--r--   1 root     other        1275 Apr 20 06:42 small.Z
                                         กลุ่ม other เมื่อดูผ่าน FTP
```

ปรับแต่งแฟ้มระบบแฟ้ม ftphosts

ในบางกรณีผู้ดูแลเซิร์ฟเวอร์อาจไม่ต้องการให้บริการกับโฮสต์บางโฮสต์ที่มักสร้างปัญหาหรือก่อภาระแก่ระบบมาก ผู้ดูแลระบบสามารถกำหนดให้เซิร์ฟเวอร์ปฏิเสธการขอใช้บริการได้เป็นรายโฮสต์โดยกำหนดไว้ในแฟ้ม */etc/ftphosts* ตัวอย่าง เช่น

```
deny    ftp          *.geo.com
deny    anonymous    *.geo.com
```

คำสั่งข้างต้นกำหนดให้เซิร์ฟเวอร์ปฏิเสธการติดต่อจากผู้ใช้ทั้ง *anonymous* และ *ftp* ที่ขอบริการมาจากโฮสต์ *prob.geo.com* โปรดสังเกตว่าการปฏิเสธต้องทำทั้งชื่อบัญชี *ftp* และ *anonymous* เนื่องจากเซิร์ฟเวอร์จะถือว่าทั้งสองบัญชีนี้ชื่อต่างกัน การปฏิเสธโฮสต์อาจใช้เครื่องหมายไวล์การ์ดได้ คำสั่งต่อไปนี้ปฏิเสธการใช้บริการจากทุกโฮสต์ใน *geo.com* และทุกไคลเอนต์จากแอดเดรส 157.109.

```
deny    ftp          *.geo.com 157.109.*
deny    anonymous    *.geo.com 157.109.*
```

นอกจากนี้ยังสามารถใช้คำสั่ง allow เพื่อกำหนดว่าจะอนุญาตให้โฮสต์ใดเข้าใช้บริการได้ ตัวอย่างเช่น

```
allow   ftp          *.lion.com
allow   anonymous    *.lion.com
```

ในกรณีที่ต้องการให้ใช้ FTP เฉพาะภายในเครือข่ายใดเครือข่ายหนึ่งเท่านั้น ตัวอย่างเช่น ในโดเมน *net.in.th* และแอดเดรส 157.109 สามารถกำหนดดังตัวอย่างต่อไปนี้

```
# more /etc/ftphosts
allow   *        *.net.in.th
allow   *        157.109.*
deny    *   *
```

หมายเหตุ : ทั้ง allow และ deny สามารถใช้งานร่วมกันได้แต่ต้องจัดลำดับให้ถูกต้อง เช่นตัวอย่างต่อไปนี้จัดคำสั่งผิดลำดับ

```
deny    ftp          *.lion.com
allow   ftp          simba.lion.com
```

เนื่องจากคำสั่ง deny จะปฏิเสธทุกโฮสต์โดเมน lion.com ดังนั้นคำสั่ง allow จึงไม่มีผลการทำงาน นอกจากนี้การระบุด้วยชื่อโดเมนจะไม่ครอบคลุมไปถึงไอพีแอดเดรส กล่าวคือ หากโฮสต์ในโดเมน *lion.com* ใช้ไอพีแอดเดรสโดยไม่จดทะเบียนชื่อโดเมน คำสั่งข้างต้นจะไม่ครอบคลุมถึงโฮสต์นั้น ยกเว้นว่าจะกำหนดคำสั่งโดยระบุไอพีแอดเดรสเพิ่มเข้าไป

ปรับแต่งแฟ้มระบบแฟ้ม ftpaccess

การติดตั้งค่าให้กับ FTP Server โดยหลักแล้วจะกำหนดไว้ในแฟ้ม *ftpaccess* ซึ่งทำหน้าที่กำหนดการทำงานหลายส่วน ซึ่งจะอธิบายดังต่อไปนี้

การกำหนดคลาสของผู้ใช้

ผู้ดูแลระบบสามารถแบ่งผู้ใช้งานออกเป็น 3 คลาส ทั้งนี้เพื่อจำกัดสิทธิการใช้บริการตามประเภทได้ คลาสผู้ใช้ทั้ง 3 ประเภทได้แก่

real กลุ่มผู้ใช้ที่ติดต่อเข้ามาโดยใช้บัญชีที่มีอยู่บนเซิร์ฟเวอร์

anonymous กลุ่มผู้ใช้ที่ติดต่อเข้ามาโดยใช้ชื่อ *anonymous* หรือ *ftp*

guest กลุ่มผู้ใช้ที่ติดต่อเข้ามาโดยใช้ ชื่อและรหัสผ่านเฉพาะ

คำสั่งกำหนดคลาสจะมีรูปแบบคำสั่งเฉพาะและมีความหมายดังนี้

classname	ชื่อคลาสที่ตั้งขึ้นเพื่อระบุชื่อคลาสผู้ใช้ (เช่น local, remote)
type	ชนิดของคลาสผู้ใช้ เช่น real, anonymous, guest
address	แอดเดรสของไคลเอนต์ โดยปกติจะใช้ชื่อโดเมน

คำสั่งผู้ขอเข้าใช้บริการจะต้องเข้ากับคลาสใดคลาสหนึ่งที่กำหนดไว้ข้างต้น มิฉะนั้นจะไม่สามารถเข้าใช้บริการได้ ขอให้พิจารณาคำสั่งต่อไปนี้

```
class   local        real        *.net.in.th
class   guests       guest       *.ac.th *.go.th *.or.th
class   anonymouses  anonymous   *
```

ตัวอย่างข้างต้นกำหนดให้ผู้ใช้บริการโดยใช้บัญชีปกติสามารถใช้ FTP ได้แต่ต้องใช้จากไคลเอนต์ที่มีโดเมน net.in.th เท่านั้น กฎข้อถัดมากำหนดว่าผู้ใช้บัญชีในกลุ่ม guest จะต้องมาจากโดเมนซึ่งลงท้ายด้วย ac.th, go.th และ or.th เท่านั้น ส่วนกฎข้อสุดท้ายกำหนดให้ผู้ใช้งานในกลุ่ม anonymous สามารถเข้าใช้บริการจากที่ใดก็ได้

การปฏิเสธการติดต่อ

หากต้องการให้เซิร์ฟเวอร์ปฏิเสธการเข้าใช้บริการจากเครื่องใดก็ตาม ให้ใช้คำสั่ง deny ในแฟ้ม ftpaccess ดังตัวอย่างต่อไปนี้

address	แอดเดรสของไคลเอนต์ โดยปกติจะใช้ชื่อโดเมน
message	แฟ้มข้อความบอกปฏิเสธการติดต่อ

ตัวอย่างเช่น คำสั่งต่อไปนี้จะปฏิเสธการเชื่อมต่อจากโดเมน co.th และแสดงข้อความปฏิเสธโดยนำข้อมูลมาจากแฟ้ม /etc/ftpmsgs/msg.denied (ต้องสร้างแฟ้มนี้ขึ้นเอง)

```
deny    *.co.th /etc/ftpmsgs/msg.denied
```

ข้อความที่ปรากฏบนจอภาพของผู้ใช้บริการจะมีลักษณะดังตัวอย่างต่อไปนี้

```
$ ftp ftp.net.in.th
Connected to ftp.net.in.th.
220 omega.net.in.th. FTP server (Version wu-2.4.2(1) Mon Jan 18
19:19:31 EST 1999) ready.
User (ftp.net.in.th:(none)): ftp
530-Sorry, you've been denied to access this server
530-This server is served only for client inside net.in.th domain.
530-ftp access denied.
Connection closed by remote host.
```

จำกัดเวลาเข้าใช้

ผู้ดูแลระบบสามารถสามารถจำกัดจำนวนการติดต่อจากผู้ใช้ได้ตามระดับที่ต่างกันโดยใช้คำสั่ง limit ตามฟอร์แมตต่อไปนี้

class	ชื่อคลาสที่ต้องการจำกัดการติดต่อ
number	จำนวนผู้ใช้สูงสุดที่มีได้ในคลาสที่กำหนด
times	วันและเวลาที่ให้บริการ โดยใช้ตัวย่อสองตัวแรกของวันและเวลาตามรูปแบบ 24 ชั่วโมง เช่น SaSu หมายถึง เสาร์ และอาทิตย์, Sa 900-1700 หมายถึง วันเสาร์ ระหว่าง 9.00 น. ถึง 17.00 น. หากใช้ Any หมายถึง ตลอดเวลา
message	แฟ้มข้อความบอกปฏิเสธการติดต่อ

ต่อไปนี้เป็นตัวอย่างการใช้คำสั่งให้มีบริการบัญชีนิรนามได้มากที่สุด 100 คน โดยไม่จำกัดเวลา

```
limit   anonymouses 100 Any /etc/ftpmsgs/msg.toomany
```

ตัวอย่างคำสั่งต่อไปนี้ให้มีบริการบัญชีนิรนามได้มากที่สุด 200 คน ในวันเสาร์และอาทิตย์

```
limit   anonymouses 200 SaSu
```

จำกัดการล็อกอินผิดพลาด

ผู้ดูแลระบบสามารถกำหนดจำนวนครั้งที่ล็อกอินผิดพลาดก่อนที่จะตัดการเชื่อมต่อโดยใช้คำสั่ง loginfails เช่น คำสั่งต่อไปนี้อนุญาตให้ล็อกอินผิดพลาดได้เพียง 3 ครั้ง และตัดการเชื่อมต่อ

```
loginfails 3
```

UNIX

กำหนดรูปแบบรหัสผ่าน

ผู้ดูแลระบบสามารถกำหนดรูปแบบการป้อนรหัสผ่านตามฟอร์แมตต่อไปนี้

Passwd-check non|trivial|rfc822 enforce|warn

passwd-check	คำสั่งกำหนดรูปแบบรหัสผ่าน
non|trivial|rfc822	Non = ไม่ตรวจสอบรหัสผ่าน Trivial =ตรวจว่ารหัสผ่านอย่างน้อยต้องมีอักษร "@" Rfc822 = ตรวจว่าแอดเดรสต้องมีทั้งชื่อบัญชีและโดเมน
enforce|warn	Enforce = ตัดการเชื่อมต่อหากป้อนรหัสผ่านผิดแบบ Wan = ส่งข้อความเตือนหากป้อนรหัสผ่านผิดแบบ

การบันทึกข้อมูลการเชื่อมต่อ

wu-ftp มีคำสั่งให้ผู้ดูแลเซิร์ฟเวอร์ใช้บันทึกว่ามีผู้ใดขอเข้าใช้บริการ FTP บ้าง โดยใช้คำสั่ง log ดังตัวอย่างต่อไปนี้

```
log transfer anonymous inbound, outbound
```

คำสั่งข้างต้นจะบันทึกการถ่ายโอนข้อมูลจากผู้ใช้ *anonymous* โดยบันทึกทั้งการถ่ายโอนเข้าเซิร์ฟเวอร์ (inbound) และการถ่ายโอนจากเซิร์ฟเวอร์ (outbound) หากต้องการบันทึกเพียงทิศทางใดทิศทางหนึ่งให้ตัดส่วนที่ไม่ต้องการออกจากคำสั่ง การถ่ายโอนข้อมูลแฟ้มบรรจุจะเก็บบันทึกภายใต้การทำงานของ **syslog** โดยปกติแล้วจะอยู่ใน */var/log/xferlog*

บัญชีห้ามใช้งาน

ftpd ยังใช้งานแฟ้มอีกแฟ้มหนึ่งคือ */etc/ftpusers* ผู้ที่มีรายชื่อในแฟ้มนี้จะไม่สามารถล็อกอินเข้าใช้บริการ FTP ได้ โดยปกติแล้วแฟ้มนี้จะบรรจุรายชื่อบัญชีระบบเพื่อป้องกันไม่ให้ใช้ FTP จากบัญชีระบบเหล่านี้ ตัวอย่างของแฟ้มนี้ได้แก่

/etc/ftpusers

```
root
bin
daemon
adm
lp
sync
shutdown
halt
mail
news
uucp
operator
games
nobody
```

โปรแกรมช่วยงาน FTP ftpcount และ ftpwho

ทำหน้าที่รายงานจำนวนผู้ใช้บริการแยกตามคลาสที่กำหนดไว้ใน *ftpaccess* รวมทั้งแสดงจำนวนผู้ใช้สูงสุดที่กำหนดไว้ ดังตัวอย่างการใช้งานต่อไปนี้

```
# ftpcount
Service class local          - 2 users (no maximum)
Service class anonymouses    - 2 users (100 maximum)
```

ส่วน **ftpwho** ทำหน้าที่คล้ายกับ **ftpcount** แต่จะแสดงถึงข้อมูลแต่ละเซสชันที่ใช้บริการเป็นราย ๆ แยกตามคลาสที่กำหนด ดังตัวอย่างการใช้งานต่อไปนี้

```
# ftpwho
Service class local:
 5610  ?  S  0:00 ftpd: cc1.net.in.th: pom: IDLE
 5607  ?  S  0:00 ftpd: www.net.in.th: pop: LIST -al
 -     2 users (no maximum)
Service class anonymouses:
 5604  ?  S  0:00 ftpd: omega.net.in.th: anonymous/
              oop@net.in.th: IDLE
 5601  ?  S  0:00 ftpd: omega.net.in.th: anonymous/ouis@:
              IDLE
 -     2  users (100 maximum)
```

สรุป

หากต้องการสร้างระบบรักษาความปลอดภัยให้แก่ระบบคอมพิวเตอร์ไม่ว่าจะเป็นการติดตั้งหรือปรับแต่งระบบให้กับบริการใด ๆ ก็ตาม สิ่งที่ควรคำนึงถึงมากที่สุดคือ บริการเหล่านั้นมีจุดอ่อนทั้งในแง่ของซอฟต์แวร์หรือส่วนที่เกี่ยวข้องกับระบบปฏิบัติการหรือไม่ หากพบว่ามีผู้ดูแลระบบควรแก้ไขโดยด่วน และถ้าไม่มั่นใจว่าจะปลอดภัยหรือไม่ ก็ควรระงับการให้บริการนั้นไว้ชั่วคราว แล้วปรับตั้งระบบใหม่ หากไม่สามารถสร้างความปลอดภัยให้แก่ระบบได้แล้วก็ควรพิจารณาดูผลเสียที่จะเกิดขึ้นว่าคุ้มค่ากับการให้บริการหรือไม่ ในกรณีที่ไม่คุ้มค่าก็ควรปิดการให้บริการนั้นเพื่อหลีกเลี่ยงความเสียหายในอนาคตที่อาจจะเกิดขึ้นได้

■ ไพโรจน์ ไววานิชกิจ

Shell Programming กับ UNIX ตอนที่ 9

เรื่องของการใช้คอมไพเลอร์ภาษาซี ในการพัฒนาโปรแกรมใช้งานบนระบบปฏิบัติการ UNIX

จากไปกับเรื่องของการเขียนโปรแกรมเชลล์สคริปต์บนเชลล์แบบบอร์นและเชลล์แบบซี สำหรับบทความในตอนนี้ผู้เขียนจะขอนำเสนอเรื่องของการใช้คอมไพเลอร์ภาษาซี ในการพัฒนาโปรแกรมใช้งานบนระบบปฏิบัติการ UNIX โดยจะมีการกล่าวถึงโปรแกรมสำคัญที่ได้รับความนิยมใช้ในการพัฒนาที่มีชื่อว่า **make** สำหรับบทความในตอนต่อ ๆ ไปเป็นเรื่องของระบบบริหารซอร์สโค้ด (Source Code Management System หรือ SCCS) ซึ่งจะถือเป็นการปิดท้ายบทความชุด Shell Programming กับ UNIX

เหตุผลสำคัญที่ทำให้ระบบปฏิบัติการ UNIX มีสภาพแวดล้อมที่เหมาะเป็นอย่างยิ่งสำหรับการพัฒนาโปรแกรมภาษาซีก็คือ การที่ภาษาซีสามารถขอเข้าใช้บริการในระบบปฏิบัติการประเภทต่าง ๆ ได้โดยง่าย โดยทั่วไปการขอใช้บริการต่าง ๆ จากเครื่องคอมพิวเตอร์จะต้องกระทำโดยร้องขอต่อระบบปฏิบัติการผ่านทาง System Call ซึ่งเป็นส่วนการทำงานที่มีหน้าที่ให้บริการแก่ผู้ใช้งาน ซึ่งส่วนดังกล่าวก็ถูกเขียนขึ้นด้วยภาษาซี หน้าที่โดยทั่วไปของ System Call ได้แก่การให้บริการสร้างไฟล์ อ่านและเขียนไฟล์ แสดงข้อมูลต่าง ๆ เกี่ยวกับไฟล์ จองหน่วยความจำ และส่งสัญญาณควบคุมไปให้กับโปรเซส (process) ต่าง ๆ ทุกครั้งที่มีการเขียนโปรแกรมภาษาซีบนระบบปฏิบัติการ UNIX ผู้พัฒนาโปรแกรมสามารถเรียกใช้ความสามารถของ System Call ได้ในลักษณะเดียวกับการใช้ฟังก์ชัน (function) เมื่อเขียนโปรแกรมภาษาซีทั่วไป

นอกจากนี้ในระบบปฏิบัติการ UNIX ยังมีไลบรารี (Library) ซึ่งเก็บรวบรวมฟังก์ชันพิเศษต่าง ๆ ที่สนับสนุนการเขียนโปรแกรมภาษาซี ฟังก์ชันส่วนใหญ่ที่มีอยู่ในไลบรารีสามารถติดต่อกับบริการทั่วไปของระบบปฏิบัติการ UNIX โดยผ่านทาง System Call อันมีผลทำให้ผู้พัฒนาโปรแกรมสามารถออกแบบโปรแกรมให้ทำงานตรงกับความต้องการของตนเองและมีประสิทธิภาพในการทำงานสูงสุด สำหรับฟังก์ชันในกลุ่มที่เหลือจะทำหน้าที่พิเศษอื่น ๆ เช่น ฟังก์ชันที่ใช้ในการคำนวณทางคณิตศาสตร์ เป็นต้น บทความในตอนนี้จะกล่าวถึงเฉพาะกระบวนการที่เกี่ยวข้องกับการสร้างและคอมไพล์โปรแกรมภาษาซีเท่านั้น ผู้อ่านที่สนใจในเรื่องของการเขียนโปรแกรมภาษาซีสามารถค้นคว้าได้จากตำราในท้องตลาดทั่วไป

UNIX

การสร้างโปรแกรมภาษาซี

ผู้พัฒนาโปรแกรมสามารถใช้โปรแกรมอีดิเตอร์ต่าง ๆ เช่น โปรแกรม vi ในการเขียนโปรแกรมภาษาซีของตนขึ้นมา ทั้งนี้ต้องไม่ลืมว่าในการตั้งชื่อไฟล์โปรแกรมภาษาซีของตนนั้นจำเป็นจะต้องกำหนดนามสกุลให้ลงท้ายด้วย ".c" เสมอ ทั้งนี้เพราะโปรแกรมคอมไพเลอร์จะทำงานกับโปรแกรมภาษาซีที่มีนามสกุลลงท้ายด้วยตัวอักษรดังกล่าวเท่านั้น สำหรับการเขียนโปรแกรมนั้นก็ไม่แตกต่างไปจากการเขียนเชลล์สคริปต์แบบซีหรือการเขียนบันทึกรายงานทั่วไป ทั้งนี้เพราะโปรแกรมอีดิเตอร์ไม่สามารถเข้าใจได้ว่าขณะนี้ผู้เขียนโปรแกรมกำลังเขียนโปรแกรมภาษาซีอยู่หรือไม่ ดังนั้นผู้พัฒนาโปรแกรมจึงจำเป็นต้องเขียนโปรแกรมตามรูปแบบโครงสร้างของภาษาซีโดยเคร่งครัด เพื่อให้คอมไพเลอร์สามารถทำงานได้โดยถูกต้อง ตัวอย่างต่อไปนี้เป็นซอร์สโค้ดภาษาซีซึ่งแสดงโครงสร้างตามหลักของภาษาซี โปรแกรมดังกล่าวจะทำหน้าที่รับข้อความจากอินพุตมาตรฐาน (เช่น คีย์บอร์ด) แล้วทำการแปลงเครื่องหมาย TAB ที่ได้รับไปเป็นอักขระช่องว่าง (Space) จำนวนตามที่กำหนด สองบรรทัดแรกของโปรแกรมจะเป็นข้อความแนะนำการทำงานของโปรแกรม ให้สังเกตว่าประโยคข้อความถูกปิดหัวท้ายด้วยเครื่องหมาย /* และ */ ซึ่งคอมไพเลอร์จะไม่นำมาประมวลผลแต่อย่างใด

```
$ cat tabs.c
<1>     /* Convert tabs in standard input to space in                    */
<2>     /* standard output while maintaining columns                     */

<3>     #include    <stdio.h>
<4>     #define     TABSIZE     8

<5>     main ( )
<6>     {
<7>     int c;                          /* character read from stdin     */
<8>     int posn = 0;                   /* column position of character  */
<9>     int inc;                        /* column increment to tab stop  */

<10>    while ((c = getchar()) != EOF)
<11>            switch (c)
<12>            {
<13>            case '\t':                                  /* c is a tab  */
<14>                    inc = findstop (posn);
<15>                    posn += inc;
<16>                    for (; inc > 0; inc--)
<17>                            putchar (' ');
<18>                    break;
<19>            case '\n':                                  /* c is a newline */
<20>                    putchar (c);
<21>                    posn = 0;
<22>                    break;
<23>            default :                                   /* c is anything else */
<24>                    putchar (c);
<25>                    posn++;
<26>                    break;
<27>            }
<28>    }

<29>    /* computs size of increment to next tab stop                    */

<30>    findstop (col)
<31>    int col;                                            /* column position of tab character     */
<32>    {
<33>    return (TABSIZE - (col % TABSIZE));
<34>    }
```

บรรทัดที่ 3 และ 4 เป็นส่วนที่เรียกว่า Preprocessor directive ซึ่งเป็นการกำหนดค่าเริ่มต้นสำหรับให้คอมไพเลอร์นำไปเตรียมการ และจะเป็นข้อมูลที่ถูกนำไปใช้ตลอดการทำงานของโปรแกรม โดยทั่วไปจะเป็นการกำหนด header file ที่ต้องการเรียกใช้ รวมถึงการกำหนดค่าคงที่ให้กับตัวแปรคงที่ตามแต่ที่ผู้เขียนโปรแกรมต้องการ การกำหนดค่าคงที่จะใช้ข้อความ **#define** ซึ่งถือเป็น Preprocessor directive ชนิดหนึ่ง ตามที่แสดงในโปรแกรม **tab.c** จะเห็นว่ามีการใช้ข้อความ **#define** กำหนดค่าคงที่กับตัวแปร TABSIZE ให้เท่ากับ 8 ซึ่งค่าดังกล่าวจะถูกนำไปใช้งานภายในโปรแกรมเพื่อกำหนดระยะห่างระหว่างตัวอักษรที่ถูกคั่นด้วยเครื่องหมาย TAB โดยทั่วไปมักนิยมตั้งชื่อข้อความที่ใช้แทนค่าคงที่ด้วยอักษรตัวพิมพ์ใหญ่

การกำหนดค่าคงที่โดยผ่านทาง Preprocessor directive มีประโยชน์ต่อผู้พัฒนาโปรแกรมมาก เนื่องจากหากต้องการเปลี่ยนค่าคงที่ก็สามารถแก้ไขได้ที่จุดเดียว แทนที่จะเป็นการเขียนค่าดังกล่าวเป็นตัวเลขไว้ตามส่วนต่าง ๆ ของโปรแกรม นอกจากนั้นผู้เขียนโปรแกรมยังสามารถใช้ข้อความ **#define** ในการสร้างมาโครเงื่อนไข ซึ่งคล้ายกับการสร้างฟังก์ชัน ลองดูการกำหนดค่าดังต่อไปนี้

```
#define     numeric(n)   (n >= '0' && n <= '9')
```

ผู้เขียนโปรแกรมสามารถเรียกใช้มาโคร numeric(n) ในโปรแกรมของตนได้ในรูปแบบเดียวกับการใช้ฟังก์ชันโดยทั่วไป อย่างไรก็ตาม มาโครก็ยังไม่ใช่ฟังก์ชันที่แท้จริง ทั้งนี้ Preprocessor จะทำการแทนที่มาโครทั้งหมดด้วยโปรแกรมภาษาซีแทนที่จะเป็นการเรียกใช้ฟังก์ชันเหมือนดังในกรณีของฟังก์ชันจริง

ในกรณีที่มีการกำหนดค่าคงที่และมาโครจำนวนมากสำหรับใช้งานในโปรแกรมหลาย ๆ ชุด ผู้พัฒนาโปรแกรมมักนิยมรวมระบุค่าคงที่และมาโครเหล่านั้นในรูปของ Preprocessor directive ไว้ด้วยกันในไฟล์ข้อความแยกต่างหาก เรียกชื่อไฟล์ดังกล่าวว่า header file หรือ include file ซึ่งแนวคิดในการสร้าง header file จะเหมาะสำหรับการนำค่าคงที่และมาโครมาบางตัวมาใช้งานบ่อย ๆ ในโปรแกรมต่าง ๆ กัน ภาษาซีไม่มีข้อกำหนดตายตัวใด ๆ ในการตั้งชื่อ header file แต่ที่ผู้เขียนโปรแกรมนิยมปฏิบัติกันก็คือการตั้งชื่อไฟล์ให้มีนามสกุลลงท้ายด้วย ".h" ชื่อของ header file ที่ต้องการใช้งานจะถูกแสดงไว้ในส่วนของ Preprocessor directive ของแต่ละโปรแกรม ตัวอย่างโปรแกรม **tab.c** จะมีการเรียกใช้มาโครชื่อ **get char** และ **putchar** ซึ่งเป็นมาโครมาตรฐานที่มีการกำหนดใน header file มาตรฐานในภาษาซีที่มีชื่อว่า **stdio.h**

เครื่องหมาย "< >" ที่ล้อมรอบชื่อ header file ในโปรแกรม **tab.c** จะทำหน้าที่แจ้งให้คอมไพเลอร์ภาษาซีมองหาไฟล์ดังกล่าวในไดเรคทอรีมาตรฐาน **/usr/include** ในกรณีที่ต้องการเรียกใช้ header file จากไดเรคทอรีอื่นจะต้องใช้เครื่องหมาย " " ร่วมกับการกำหนดพาธสมบูรณ์ที่แสดงตำแหน่งของ header file ที่ต้องการ ตัวอย่างต่อไปนี้เป็นการอ้างถึง header file ที่อยู่ในไดเรคทอรีใช้งานของผู้เขียน

```
#include "/home/pairoj/cprogs/legs.h"
```

ส่วนประกอบสำคัญที่จะต้องมีในโปรแกรมภาษาซีทุกโปรแกรมก็คือฟังก์ชันที่มีชื่อว่า main ( ) แม้ในการเขียนโปรแกรมจะมีการกำหนดฟังก์ชันต่าง ๆ ไว้มากเพียงใด แต่ผู้เขียนโปรแกรมจะต้องจำไว้เสมอว่าคอมไพเลอร์จะเริ่มต้นการทำงานตามกระบวนการคำสั่งที่ระบุไว้ในฟังก์ชัน main ( ) เสมอ นั่นคือบรรดาฟังก์ชันต่าง ๆ ที่ถูกระบุไว้ในโปรแกรมจะถูกเรียกใช้ผ่านทางฟังก์ชัน main ( ) ซึ่งเป็นฟังก์ชันหลักตามเงื่อนไขที่ผู้เขียนโปรแกรมได้กำหนดไว้ ผู้เขียนโปรแกรมสามารถเขียนฟังก์ชัน main ( ) ให้อยู่ในตำแหน่งใดของโปรแกรมก็ได้ สำหรับโปรแกรม **tabs.c** นั้นมีการเรียกใช้ฟังก์ชัน **findstop** สำหรับคำนวณหาระยะระหว่างตัวอักษรที่ถูกคั่นด้วยเครื่องหมาย TAB แม้จะเห็นว่ารายละเอียดของฟังก์ชันจะดูสั้นและสามารถรวมไว้ในฟังก์ชัน main ( ) แต่ในทางปฏิบัติก็ควรจะแยกออกมาเป็นฟังก์ชันต่างหาก เนื่องจากผู้เขียนโปรแกรมสามารถเรียกใช้ฟังก์ชันดังกล่าวซ้ำได้โดยเพียงอ้างชื่อฟังก์ชันเท่านั้น นอกจากนั้นยังทำให้การตรวจสอบการทำงานของโปรแกรมเป็นไปได้โดยง่าย

หากโปรแกรมที่เขียนมีความยาวมากและมีการเรียกใช้ฟังก์ชันเป็นจำนวนมาก ผู้เขียนโปรแกรมอาจต้องการแยกโปรแกรมดังกล่าวออกเป็นไฟล์ย่อย ๆ ซึ่งก็เป็นเรื่องที่เป็นไปได้ อย่างไรก็ตาม ฟังก์ชันต่าง ๆ ที่มีการเรียกใช้ก็จะต้องถูกระบุไว้อย่างครบถ้วนในไฟล์แต่ละไฟล์ ทั้งนี้ต้องไม่ลืมการระบุเรียกใช้ header file ที่ต้องการไว้ในส่วน Preprocessor directive ของโปรแกรมย่อยแต่ละไฟล์ และต้องตั้งชื่อแต่ละไฟล์ให้มีนามสกุลลงท้ายด้วย ".c"

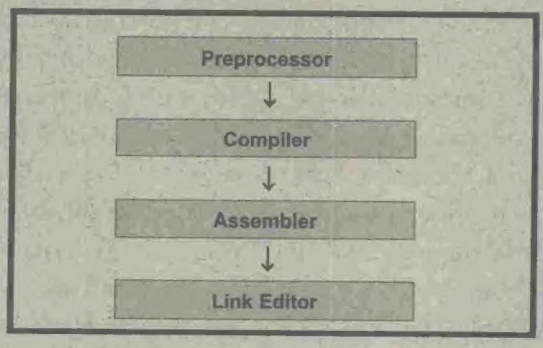
UNIX

การคอมไพล์โปรแกรมภาษาซี

การคอมไพล์โปรแกรมภาษาซีเป็นกระบวนการในการแปลงให้ซอร์สโค้ดที่เขียนขึ้นสามารถเรียกใช้งานได้ ในกรณีของระบบปฏิบัติการ UNIX ผู้เขียนโปรแกรมสามารถเรียกใช้คำสั่ง **cc** เพื่อสร้างโปรแกรมที่สามารถเรียกใช้งานได้ ในกรณีตัวอย่างของโปรแกรม tab.c ผู้เขียนโปรแกรมสามารถทำการคอมไพล์โดยใช้คำสั่งต่อไปนี้

```
$ cc tabs.c
```

คำสั่ง **cc** จะทำการเรียกใช้งานส่วนประกอบต่าง ๆ อันได้แก่ Preprocessor, Compiler, Assembler และ Link Editor ซึ่งเป็นส่วนประกอบ 4 ชุดในกระบวนการคอมไพล์โปรแกรมภาษาซี ลำดับขั้นตอนของกระบวนการคอมไพล์แสดงในรูปที่ 1 เริ่มจาก Preprocessor ซึ่งทำหน้าที่นำค่าคงที่และมาโครต่าง ๆ ที่ถูกกำหนดไว้ในส่วนของ Preprocessor directive และใน include file ไปแทนที่ไว้ในส่วนต่าง ๆ ที่มีการเรียกใช้งานในโปรแกรม จากนั้นจึงส่งผ่านการทำงานไปยังส่วนของ Complier ซึ่งจะทำการแปลงโปรแกรมที่เขียนไว้ให้อยู่ในรูปของรหัสโปรแกรมภาษาแอสเซมบลี แล้วส่งผ่านไฟล์แอสเซมบลีที่ได้ไปให้กับส่วน Assembler ส่วนการทำงาน Assembler จะทำการสร้างรหัสภาษาเครื่อง (Machine Language) ในรูปของออบเจ็กต์ (Object) โดยแต่ละออบเจ็กต์จะแทนโปรแกรมภาษาซีแต่ละโปรแกรม ออบเจ็กต์ไฟล์จะใช้ชื่อเดียวกับชื่อไฟล์โปรแกรมภาษาซีที่ได้สร้างไว้ เพียงแต่มีการเปลี่ยนแปลงนามสกุลจาก ".c" ไปเป็น ".o" จากตัวอย่างที่ผ่านมา เมื่อมีการสั่งคอมไพล์โปรแกรม **tabs.c** แล้ว สิ่งที่จะได้กลับมาหลังจากผ่านกระบวนการ Assembler ก็คือออบเจ็กต์ไฟล์ที่มีชื่อว่า **tabs.o** อย่างไรก็ตาม หลังจากเสร็จสิ้นกระบวนการคอมไพล์ทั้งหมด ออบเจ็กต์ไฟล์จะถูกลบทิ้งไป ดังนั้นในการทำงานจริง ผู้พัฒนาโปรแกรมจึงมักไม่มีโอกาสได้เห็นไฟล์ดังกล่าว



รูปที่ 1
องค์ประกอบต่าง ๆ ที่เกี่ยวข้องกับกระบวนการคอมไพล์

กระบวนการสุดท้ายของการคอมไพล์ก็คือส่วนของ Link Editor ซึ่งจะทำหน้าที่ตรวจหาไฟล์ไลบรารี (Library) สำหรับใช้กับฟังก์ชันต่าง ๆ ที่มีการระบุในตัวโปรแกรม ทั้งนี้เพราะ header file ทำหน้าที่เพียงบอกชื่อฟังก์ชันที่มีอยู่เท่านั้น หากต้องการเรียกใช้งานจริงก็จำเป็นจะต้องอ้างถึงเนื้อหาของฟังก์ชันในไฟล์ไลบรารี นอกจากนี้ยังทำหน้าที่รวมโมดูลของออบเจ็กต์ทั้งหมดที่เกี่ยวข้องกับโปรแกรมเข้าด้วยกัน โดยปกติคอมไพเลอร์ภาษาซีจะตรวจหาจากไฟล์ไลบรารีมาตรฐานที่มีชื่อว่า **libc.a** ซึ่งรวมฟังก์ชันมาตรฐานต่าง ๆ ที่เกี่ยวข้องกับการจัดการอินพุตและเอาต์พุต หากผู้เขียนโปรแกรมต้องการให้ Link Editor ตรวจหาไฟล์ไลบรารีอื่น ๆ ก็สามารถทำได้โดยเพิ่มอาร์กิวเมนต์ **-l** ต่อท้ายคำสั่งคอมไพล์ตั้งแต่ต้น พึงระวังไว้ว่าจะต้องเพิ่มอาร์กิวเมนต์ดังกล่าวต่อท้ายชื่อไฟล์ที่ต้องการจะทำการคอมไพล์เท่านั้น ตัวอย่างต่อไปนี้เป็นการสั่งคอมไพล์โปรแกรมชื่อ **calc.c** โดยต้องการเรียกใช้ไฟล์ไลบรารีชื่อ **libm.a** ซึ่งเป็นไลบรารีที่เก็บฟังก์ชันทางด้านการคำนวณ สังเกตว่ามีการใช้ตัวอักษรตัวสุดท้ายก่อนถึงนามสกุลของไฟล์ไลบรารีมาเป็นการอ้างอิงในอาร์กิวเมนต์ ซึ่งตัวอักษร "m" ที่ปรากฏในตัวอย่างเป็นการย่อถึงไฟล์ไลบรารีชื่อ **libm.a** นั่นเอง

```
$ cc calc.c -lm
```

กระบวนการคอมไพล์สิ้นสุดลงด้วยการที่ Link Editor สร้างไฟล์ที่สามารถเรียกใช้งานได้ (Executable file) ซึ่งมีชื่อว่า **a.out** ในกรณีที่การคอมไพล์เกี่ยวข้องกับออบเจ็กต์ไฟล์เพียงไฟล์เดียว คำสั่ง **cc** จะทำการลบออบเจ็กต์ไฟล์นั้นในทันทีที่เสร็จสิ้นกระบวนการดังได้กล่าวมาแล้ว ตัวอย่างต่อไปเป็นการคอมไพล์ที่เกี่ยวข้องกับออบเจ็กต์ไฟล์จำนวนหลายไฟล์ ซึ่งโดยปกติคำสั่ง **cc** จะไม่ทำการลบไฟล์เหล่านี้ ผู้เขียนโปรแกรมควรใช้อาร์กิวเมนต์ **-O** ต่อท้ายคำสั่ง **cc** ซึ่งเป็นการเรียกใช้คอมไพเลอร์ภาษาซีชนิด Optimizer โดยคอมไพเลอร์ดังกล่าวจะทำให้การเรียกใช้โปรแกรมเป็นไปได้รวดเร็วขึ้น

```
$ cc -0 ledger.c acctspay.c acctsrec.c
$ ls
a.out       acctspay.o          acctsrec.o    ledger.o
acctspay.c  acctsrec.c          ledger.c
```

การเรียกใช้โปรแกรมที่ผ่านการคอมไพล์แล้วก็ทำได้ในลักษณะเดียวกันกับการเรียกใช้เชลล์สคริปต์และโปรแกรมทั่ว ๆ ไป ทั้งนี้โดยการพิมพ์ชื่อโปรแกรมตามด้วยการกดปุ่ม **ENTER** สำหรับโปรแกรมตัวอย่าง **tabs.c** ที่ได้กล่าวถึงตั้งแต่ต้น หลังจากผ่านกระบวนการคอมไพล์มาแล้วก็จะมีชื่อว่า a.out ดังที่ทราบแล้วโปรแกรมดังกล่าวจะรอรับข้อมูลที่ป้อนผ่านเข้ามาทางอินพุตมาตรฐาน ผู้เขียนโปรแกรมจึงสามารถเรียกใช้งานโปรแกรมพร้อมกับป้อนข้อมูลที่ต้องการ ซึ่งเพื่อเป็นความสะดวกจึงมักจะป้อนจากไฟล์ข้อความที่ต้องการทำงานด้วยเข้าไป ดังตัวอย่างต่อไปนี้

```
$ a.out < mymemo
```

อย่างไรก็ตาม เนื่องจากคอมไพเลอร์จะสร้างไฟล์ผลลัพธ์ในชื่อของ **a.out** ในทุกครั้ง โอกาสที่ผู้เขียนโปรแกรมจะทำการคอมไพล์โปรแกรมอื่น ๆ แล้วเกิดการเขียนทับโปรแกรมที่มีอยู่ก่อนแล้ว ย่อมเป็นไปได้เสมอ เพื่อเป็นการหลีกเลี่ยงปัญหาดังกล่าวจึงควรทำการแก้ไขชื่อไฟล์จาก **a.out** ไปเป็นชื่อที่เหมาะสมโดยการใช้คำสั่ง **mv** อย่างไรก็ตาม ผู้เขียนโปรแกรมสามารถทำการกำหนดชื่อไฟล์ที่ต้องการได้ตั้งแต่เริ่มกระบวนการคอมไพล์ โดยเพิ่มอาร์กิวเมนต์ **-o** (อักษรตัวเล็ก) ตามด้วยชื่อไฟล์ผลลัพธ์ที่ต้องการ แล้วจึงตามต่อด้วยชื่อของไฟล์ต้นทาง ตัวอย่างต่อไปนี้เป็นการกำหนดให้คอมไพเลอร์ภาษาซีทำการสร้างไฟล์ผลลัพธ์ในชื่อว่า **accounting**

```
$ cc -o accounting ledger.c acctspay.c acctsrec.c
```

ในกระบวนการคอมไพล์โดยทั่วไป เวลาที่ใช้โดย Link Editor มักจะกินเวลานาน สมมติว่าผู้เขียนโปรแกรมพิมพ์ข้อความผิดพลาดซึ่งคอมไพเลอร์จะตรวจพบและแสดงข้อความผิดพลาดก่อนที่จะสร้างไฟล์ออบเจ็กต์ อย่างไรก็ตาม ผู้เขียนโปรแกรมก็ไม่สามารถจะหยุดการทำงานในขั้นตอนนี้เพื่อทำการแก้ไขข้อความในส่วนที่ผิดพลาดได้ในทันที แต่จะต้องรอจนกระทั่งกระบวนการคอมไพล์ในขั้นสุดท้ายคือ Link Editor เสร็จสิ้นลง หากมีความผิดพลาดที่ต้องแก้ไขเกิดขึ้นบ่อย ๆ ก็จะทำให้เสียเวลารอคอยนานมากเกินไป ดังนั้นโปรแกรมคอมไพเลอร์ภาษาซีบนระบบปฏิบัติการ UNIX จึงเสนอทางเลือกให้ผู้เขียนโปรแกรมเลือกทำเฉพาะการสร้างไฟล์ออบเจ็กต์เพียงอย่างเดียว เมื่อแน่ใจว่าไฟล์ออบเจ็กต์ไม่มีปัญหาใด ๆ แล้วก็จะสามารถผ่านไปสู่ขั้นตอนของ Link Editor เพื่อสร้างไฟล์ผลลัพธ์ออกมาในท้ายที่สุด การคอมไพล์โดยตัดกระบวนการของ Link Editor จะกระทำโดยใช้คำสั่ง cc ร่วมกับอาร์กิวเมนต์ -c ดังแสดงในตัวอย่างต่อไปนี้

```
$ cc -c ledger.c acctspay.c acctsrec.c
$ ls
acctpay.c  acctspay.o  acctsrec.c  acctsrec.o
ledger.c   ledger.o
```

หลังจากมั่นใจว่าได้มีการแก้ไขเพื่อความถูกต้องของโปรแกรมในแต่ละโมดูลแล้ว ผู้เขียนโปรแกรมสามารถสั่งการคอมไพล์เฉพาะในขั้นตอนของ Link Editor เพื่อสร้างไฟล์ผลลัพธ์ ซึ่งคอมไพเลอร์ภาษาซีเปิดโอกาสให้ผู้เขียนโปรแกรมสามารถเลือกไฟล์ทั้งที่เป็น ".c" และ ".o" ร่วมกันเพื่อสร้างไฟล์ผลลัพธ์ ในตัวอย่างต่อไปนี้เนื่องจากได้มีการสร้างไฟล์ออบเจ็กต์รอไว้เป็นที่เรียบร้อยแล้ว ผู้เขียนโปรแกรมจึงสามารถพิมพ์คำสั่งได้ดังนี้

```
$ cc -o accounting ledger.o acctspay.o acctsrec.o
```

สำหรับการเขียนโปรแกรมซึ่งต้องมีการอ้างถึงโมดูลเป็นจำนวนมาก ผู้เขียนโปรแกรมย่อมต้องการที่จะทำการคอมไพล์เฉพาะโมดูลแยกจากกัน และสามารถนำแต่ละโมดูลมาเชื่อมต่อกันในภายหลัง ซึ่งถือเป็นการประหยัดเวลาและมีความยืดหยุ่นสูงในการเขียนโปรแกรม ระบบปฏิบัติการ UNIX ก็ได้มีการจัดเตรียมคำสั่ง make เพื่อใช้อำนวยความสะดวกในการจัดการ ผู้เขียนจะได้กล่าวถึงคำสั่งดังกล่าวในบทความตอนต่อไป

การดีบักโปรแกรมภาษาซี

คอมไพเลอร์ภาษาซีจัดว่าเป็นโปรแกรมที่เปิดกว้างสำหรับการเขียนโปรแกรมในหลากหลายรูปแบบ โดยเฉพาะอย่างยิ่งโปรแกรม cc ค่อนข้างจะทำงานไปตามแนวคิดที่ว่า "หากผู้ใช้งานป้อนคำสั่งอย่างไร ก็จะทำไปตามคำสั่งนั้น" และการแจ้งข้อความผิดพลาดที่อาจเกิดขึ้นก็ไม่หลากหลายและครอบคลุมทุกสถานการณ์ การตรวจสอบความผิดพลาดของซอร์สโค้ดที่ป้อนเข้าไปนั้นจะอาศัยการเปรียบเทียบให้เป็นไปตามหลักไวยากรณ์และโครงสร้างของภาษาซีมาตรฐานเท่านั้น บ่อยครั้งที่ผู้เขียนโปรแกรม

UNIX

ต้องพบกับปัญหาที่เกิดขึ้นกับโปรแกรมในภายหลัง โดยที่คอมไพเลอร์มิได้แจ้งแสดงความผิดพลาดออกมาแต่อย่างใด อย่างไรก็ตาม การทำงานในลักษณะดังกล่าวก็มิได้ถือว่าเป็นจุดบกพร่องร้ายแรงแต่อย่างใด เมื่อแลกกับการที่โปรแกรม cc ให้อิสระกับการควบคุมและแก้ไขการคอมไพล์ได้อย่างมีประสิทธิภาพ เรื่องที่ต้องพิจารณาคือแนวทางในการตรวจสอบปัญหาของโปรแกรม หรือการดีบักโปรแกรม (Debugging) ซึ่งผู้เขียนโปรแกรมต้องหาเทคนิคที่เหมาะสมมาประยุกต์ใช้งาน

วิธีการง่าย ๆ ที่ผู้เขียนโปรแกรมสามารถใช้ดีบักโปรแกรมภาษาซีก็คือการเพิ่มคำสั่งพิมพ์ข้อความแจ้งเตือนตามจุดต่าง ๆ ที่สำคัญในซอร์สโค้ด ซึ่งเป็นการแจ้งเตือนให้กับผู้ใช้งานถึงปัญหาที่อาจเกิดขึ้นจากการเขียนโปรแกรมและหลุดรอดจากการตรวจสอบของคอมไพเลอร์ ตัวอย่างเช่น ในกรณีของโปรแกรม **tabs.c** ในตอนต้น หากผู้เขียนโปรแกรมต้องการตรวจสอบปัญหาที่อาจเกิดขึ้นขณะเรียกใช้งานโปรแกรม ก็สามารถเพิ่มข้อความโดยใช้คำสั่ง **fprintf ( )** เพื่อให้มีการแสดงข้อความตามจุดต่าง ๆ ที่โปรแกรมได้ทำงานผ่านไป หากมีปัญหาใดเกิดขึ้นในระหว่างการทำงานจริง ผู้เขียนโปรแกรมก็จะสามารถทราบได้ว่าปัญหาเกิดขึ้นที่จุดใดในตัวโปรแกรม โดยการสังเกตจากตำแหน่งของข้อความแจ้งการทำงานในขณะที่เกิดปัญหาขึ้น ผู้เขียนขอตัดตอนแสดงโปรแกรม **tabs.c** เฉพาะที่มีการเพิ่มข้อความแสดงการทำงานดังนี้

```
case '\t':              /* c is a tab */
        fprintf (stderr, "before call to findstop, posn is %d\n", posn);
        inc = findstop (posn);
        fprintf (stderr, "after call to findstop, posn is %d\n", posn);
        posn += inc;
        for (; inc > 0; inc--)
                putchar (' ');
        break;
case '\n':              /* c is a newline */
        fprintf (stderr, "got a new line\n");
        putchar (c);
        posn = 0;
        break;
default:                /* c is anything else */
        fprintf (stderr, "got another character\n");
        putchar (c);
        posn++;
        break;
```

ข้อความที่ขีดเส้นใต้คือคำสั่งที่เพิ่มขึ้นมา เนื่องจากมีการกำหนดทิศทางการแสดงผลของข้อความเหล่านี้ไปยังหน่วยข้อความผิดพลาดมาตรฐาน (Standard Error) ดังนั้นผู้ใช้งานย่อมสามารถเปลี่ยนทิศทางการแสดงผลเพื่อแยกเอาข้อความแจ้งผลเหล่านี้ออกจากเอาต์พุตที่แท้จริงของตัวโปรแกรมได้ ตัวอย่างต่อไปนี้เป็นผลลัพธ์ที่ได้จากการทดลองใช้งานโปรแกรมดังกล่าว โดยให้รับอินพุตมาจากไฟล์ข้อความชื่อ testtabs และส่งผลลัพธ์เฉพาะที่ได้จากการทำงานของโปรแกรมไปยังไฟล์ชื่อ **testspaces**

```
$ cat testtabs          <— ไฟล์ข้อความที่นำมาทดสอบ
xyz<TAB>abc
$ tabs < testtabs > testspaces
got another character                   <— ตัวอักษร x
got another character                   <— ตัวอักษร y
got another character                   <— ตัวอักษร z
before call to findstop, posn is 3
after call to findstop, posn is 5
got another character                   <— ตัวอักษร a
got another character                   <— ตัวอักษร b
got another character                   <— ตัวอักษร c
got a new line
$ cat testspaces
xyz     abc
$
```

เทคนิคดังกล่าวเหมาะสำหรับการโปรแกรมที่ไม่ยาวมากนัก และผู้เขียนโปรแกรมค่อนข้างจะทราบขั้นตอนการทำงานของโปรแกรมเป็นอย่างดี อย่างไรก็ตาม หากต้องทำงานกับโปรแกรมขนาดใหญ่ที่มีความซับซ้อนมาก ระบบปฏิบัติการ UNIX ก็ได้มีการจัดเตรียมโปรแกรมสนับสนุนการดีบักไว้ให้กับผู้เขียนโปรแกรม โปรแกรมที่ผู้เขียนจะกล่าวถึงมีชื่อว่า **lint** เป็นโปรแกรมที่ได้รับความนิยมใช้ในการดีบักซอร์สโค้ด โดยโปรแกรมดังกล่าวจะตรวจสอบหาข้อผิดพลาดหลักรวมถึงปัญหาย่อยต่าง ๆ การทำงานของโปรแกรม **lint** จะเคร่งครัดกว่าคอมไพเลอร์ภาษาซีโดยทั่วไปมาก เนื่องจากมีการตรวจแจ้งปัญหาและข้อผิดพลาดแทบทุกประเภท ไม่ว่าจะเป็นข้อผิดพลาดที่ตรวจพบในทันทีหรือสิ่งที่สามารถก่อให้เกิดปัญหาขณะทำงาน รวมไปถึงเรื่องจุกจิกต่าง ๆ เช่น การเรียกใช้ตัวแปรก่อนที่จะมีการประกาศตัว (Declaration) การกำหนดอาร์กิวเมนต์หรือฟังก์ชันไว้แล้วไม่ได้ใช้งานและฟังก์ชันซึ่งมีการส่งค่าคืนกลับ (Return value) ที่ไม่มีระบุไว้ เป็นต้น คำสั่ง **lint** จะรายงานปัญหาที่ไม่สามารถตรวจพบได้จากคอมไพเลอร์ทั่วไป โดยเมื่อใช้งานคำสั่งดังกล่าวกับโปรแกรม tabs.c จะได้ผลลัพธ์ดังนี้

UNIX

```
$ lint tabs.c
tabs.c(27): warning: main() returns random value
to invocation environment
putc returns value which is always ignored
```

โดยทั่วไป โปรแกรมใด ๆ ก็ตามที่ทำงานเสร็จสิ้นลงโดยสมบูรณ์ควรส่งค่าคืนกลับเป็นศูนย์ ดังนั้นเพื่อเป็นการกำจัดข้อความแจ้งเตือนในบรรทัดแรกออกไป ผู้เขียนโปรแกรมจึงควรทำการกำหนดค่าคืนกลับให้กับฟังก์ชัน main ( ) ในโปรแกรม **tabs.c** ให้มีค่าเท่ากับศูนย์โดยเพิ่มคำสั่ง **return 0;** ในบรรทัดสุดท้ายก่อนจบฟังก์ชัน main ( ) สำหรับข้อความแจ้งเตือนในบรรทัดถัดมาดูค่อนข้างน่าแปลก เนื่องจากในโปรแกรม **tabs.c** มิได้มีการเรียกใช้ฟังก์ชัน **putc** แต่อย่างใด อันที่จริงแล้วในโปรแกรมมีการเรียกใช้ฟังก์ชัน **putchar** ซึ่งเป็นมาโครที่ทำการเรียกฟังก์ชัน **putc** ในไลบรารีมาตรฐานอีกต่อหนึ่ง ปัญหาที่ตรวจพบก็คือไม่มีการตรวจสอบค่าคืนกลับจากเรียกใช้มาโคร **putchar** ซึ่งหากผลจากการทำงานของ **putchar** เกิดล้มเหลวขึ้น โปรแกรมที่เรียกใช้งานย่อมไม่ทราบและส่งผลให้ข้อมูลเอาต์พุตที่ได้จากโปรแกรมเกิดความผิดพลาด

ผู้เขียนโปรแกรมโดยส่วนใหญ่มักเพิกเฉยไม่สนใจต่อข้อความแจ้งเตือนของโปรแกรม **lint** เนื่องจากการแจ้งเตือนส่วนใหญ่ค่อนข้างเป็นการเตือนถึงการรักษากฎระเบียบอันเคร่งครัดของการเขียนโปรแกรม แต่มักไม่มีผลกระทบใด ๆ ต่อการทำงานของโปรแกรม อย่างไรก็ตาม การเอาใจใส่ต่อข้อความที่ได้จากโปรแกรม lint ก็ถือเป็นเรื่องที่ดีในการฝึกทักษะการเขียนโปรแกรมที่ดีในอนาคต

นอกจากโปรแกรมสนับสนุนการดีบักเช่น **lint** แล้ว ระบบปฏิบัติการ UNIX ก็ยังมีโปรแกรมพิเศษที่ใช้ในการดีบักซึ่งมีประสิทธิภาพเหนือกว่า **lint** และคอมไพเลอร์ทั่วไปเป็นอันมาก โปรแกรมดังกล่าวได้แก่ **adb, sdb** และ **dbx** ซึ่งโปรแกรม **dbx** จัดได้ว่าเป็นโปรแกรมดีบักในระดับสูงที่เปิดโอกาสให้ผู้เขียนโปรแกรมวิเคราะห์การทำงานของโปรแกรมในระดับของโครงสร้างภาษาซีโดยตรง นอกจากนั้นผู้เขียนโปรแกรมยังสามารถเลือกดีบักโปรแกรมในระดับต่ำด้วยภาษาเครื่องได้เช่นเดียวกัน โปรแกรม **dbx** เปิดโอกาสให้ผู้เขียนโปรแกรมในการตรวจสอบและควบคุมการทำงานของโปรแกรมในแต่ละขั้นตอน โดยผู้ใช้งานสามารถหยุดและเลื่อนการทำงานไปในแต่ละบรรทัด ซึ่งในการทำงานตามคำสั่งแต่ละบรรทัดจะมีการแสดงค่าของตัวแปรและสภาพแวดล้อมของระบบปฏิบัติการไว้เป็นข้อมูลสำหรับการวิเคราะห์

โดยปกติในระบบปฏิบัติการ UNIX เมื่อมีการเรียกใช้งานโปรแกรมใด ๆ ก็ตาม หากโปรแกรมดังกล่าวมีการทำงานผิดพลาดและมีผลกระทบต่อระบบอย่างรุนแรง ระบบปฏิบัติการจะแสดงข้อความ "Segmentation violation - Core dumped" หรือข้อความอื่น ๆ ที่มีความหมายใกล้เคียงกัน พร้อมกับสร้างไฟล์ **core** ซึ่งเก็บรวบรวมข้อมูลแสดงสถานะของโปรแกรมและระบบในขณะที่เกิดความผิดพลาดขึ้น ผู้เขียนโปรแกรมสามารถเรียกใช้โปรแกรม **dbx** ในการระบุบรรทัดคำสั่งที่ทำให้เกิดข้อผิดพลาดขึ้น พร้อมกับแสดงค่าของตัวแปรต่าง ๆ ในเวลานั้นโดยการอ่านค่าจากไฟล์ **core** อย่างไรก็ตาม เนื่องจากโดยทั่วไปไฟล์ **core** จะมีขนาดค่อนข้างใหญ่ทำให้เปลืองเนื้อที่ในการจัดเก็บข้อมูล จึงควรทำการลบไฟล์ดังกล่าวทิ้งเมื่อเลิกใช้

UNIX

หากผู้เขียนโปรแกรมต้องการเรียกใช้โปรแกรมดีบักเกอร์ร่วมกับโปรแกรมที่เขียนขึ้น ในขณะที่ทำการคอมไพล์โปรแกรมด้วยคำสั่ง **cc** ก็ต้องเพิ่มอาร์กิวเมนต์ **-g** เพื่อทำให้คำสั่ง **cc** สร้างข้อมูลเพิ่มเติมสำหรับนำไปใช้กับโปรแกรมดีบักเกอร์ดังตัวอย่างที่จะกล่าวถึงต่อไป หากข้ามขั้นตอนดังกล่าวโดยไปเรียกใช้โปรแกรมดีบักเกอร์ เช่น **dbx** โดยมิได้ผ่านการเรียกใช้คำสั่งคอมไพล์ก่อน ผู้เขียนโปรแกรมก็จะได้รับข้อความแจ้งเตือนจากโปรแกรม **dbx** ซึ่งมีทางแก้คือทำการคอมไพล์โปรแกรมด้วยอาร์กิวเมนต์ -g พร้อมระบุชื่อไฟล์ที่ต้องการสร้างขึ้นใหม่เพื่อใช้งานกับโปรแกรมดีบักเกอร์ ผู้เขียนขออ้างถึงโปรแกรม **tabs.c** ซึ่งในกรณีนี้จะสร้างไฟล์สำหรับทำการดีบักภายใต้ชื่อว่า **tabs**

```
$ dbx tabs
Reading symbolic information...
Read 12 symbols
Warning: main routine not compiled with the -g option  <— ข้อความแจ้งเตือน
.
.
$ cc -g tabs.c -o tabs
```

เมื่อแน่ใจว่าได้สร้างไฟล์สำหรับใช้ประกอบการดีบักไว้เป็นที่เรียบร้อยแล้ว ผู้เขียนโปรแกรมก็จะสามารถเรียกใช้โปรแกรม **dbx** ร่วมกับชื่อไฟล์ที่ต้องการดีบัก โดยเมื่อโปรแกรม **dbx** อ่านข้อมูลในไฟล์ที่ระบุไว้แล้ว ก็จะแสดงเครื่องพรอมต์ (dbx) ขึ้นบนหน้าจอ ณ จุดนี้ ผู้ใช้งานสามารถใช้คำสั่งควบคุมต่าง ๆ ของโปรแกรม **dbx** ในการดีบักข้อมูล คำสั่งแรกที่จะกล่าวถึงคือคำสั่ง **list** ซึ่งจะทำหน้าที่แสดงเนื้อหาของโปรแกรมใน 10 บรรทัดแรกให้ปรากฏบนหน้าจอ ทั้งนี้จะข้ามการแสดงข้อความในส่วนของ Preprocessor directive เนื่องจากข้อความดังกล่าวได้ผ่านการอ่านและนำไปประมวลผลเป็นที่เรียบร้อยโดยคำสั่ง **cc** ก่อนที่จะมีการสร้างไฟล์ข้อมูลดังกล่าวแล้ว ตัวอย่างต่อไปนี้เป็นการเรียกใช้โปรแกรม **dbx** และแสดงผลลัพธ์ที่ได้จากการใช้คำสั่ง **list** ทั้งนี้ถือว่าได้มีการคอมไพล์สร้างไฟล์ข้อมูลสำหรับการดีบักชื่อว่า **tabs** ไว้ก่อนหน้านี้แล้ว

```
$ dbx tabs
Reading symbolic information...
Read 63 symbols
(dbx) list
7       int posn=0;
8       int inc;
9
10      while ((c=getchar()) != EOF)
11      switch(c)
12      {
13      case '\t':
14      inc = findstop (posn);
15      posn += inc;
16      for ( ; inc > 0; inc—)
```

คำสั่งต่อไปที่ขอกล่าวถึงคือชุดคำสั่งที่ใช้ในการตรวจสอบต่าง ๆ เริ่มจากคำสั่ง **where** ซึ่งใช้ตรวจสอบว่าฟังก์ชันใดที่กำลังทำงานอยู่ในปัจจุบัน รวมถึงให้มีการแสดงแอดเดรสของหน่วยความจำที่ฟังก์ชันดังกล่าวใช้งานอยู่ (แสดงเป็นตัวเลขฐานสิบหก) นอกจากผู้เขียนโปรแกรมยังสามารถใช้คำสั่ง **whatis** เพื่อตรวจหาการประกาศตัวแปรที่ต้องการทราบ ใช้ในการตรวจสอบว่าตัวแปรที่ผู้เขียนโปรแกรมสนใจอยู่เป็นตัวแปรประเภทใด และยังมีคำสั่ง **whereis** สำหรับใช้ระบุตำแหน่งของตัวแปรในโปรแกรม ตัวอย่างต่อไปนี้แสดงการใช้งานคำสั่งทั้งสามชุด คำสั่ง **where** แสดงผลลัพธ์ว่าขณะนี้โปรแกรมกำลังทำงานในส่วนของฟังก์ชัน main ( ) โดยใช้แอดเดรสหน่วยความจำที่ตำแหน่ง 0x2290 คำสั่ง **whatis col** ทำให้ทราบว่าตัวแปร **col** เป็นตัวแปรประเภท integer และมีการประกาศใช้งานในฟังก์ชัน **findstop** ภายในโปรแกรม **tabs** โดยทราบจากการใช้คำสั่ง **whereis col**

```
(dbx) where
main() at 0x2290
(dbx) whatis col
int col;
(dbx) whereis col
variable: 'tabs'findstop'col
```

ความสามารถประการสำคัญของโปรแกรม **dbx** ก็คือการสั่งให้โปรแกรมที่กำลังทำการดีบักอยู่ดำเนินการภายใต้เงื่อนไขต่าง ๆ ที่ผู้เขียนโปรแกรมสามารถ

UNIX

กำหนดเองได้ เช่น ผู้เขียนโปรแกรมสามารถสั่งหยุดการทำงานของโปรแกรมได้เมื่อต้องการทราบสถานะของตัวแปรบางตัว หรือมิฉะนั้นก็สามารถสั่งให้โปรแกรมหยุดการทำงานเมื่อมีการเรียกใช้ฟังก์ชันที่กำหนดไว้ ตัวอย่างต่อไปนี้เป็นการสาธิตความสามารถดังกล่าวของโปรแกรม **dbx** โดยให้ถือว่ามีการเรียกใช้โปรแกรม **dbx** โดยอ้างชื่อไฟล์ข้อมูล **tabs** และมีการใช้ไฟล์ **testtabs** เป็นข้อมูลอินพุต เมื่อโปรแกรม **tabs** หยุดทำงานในขณะที่เริ่มมีการเรียกฟังก์ชัน **findstop** ผู้เขียนโปรแกรมสามารถใช้คำสั่ง **print** เพื่อแสดงค่าตัวแปรที่ต้องการตรวจสอบ ณ ตำแหน่งที่โปรแกรมหยุดทำงาน นอกจากนี้ยังสามารถใช้คำสั่ง **where** เพื่อตรวจสอบว่ามีฟังก์ชันใดบ้างที่กำลังทำงานอยู่ และถูกเรียกใช้โดยฟังก์ชันใด ตัวอย่างแสดงว่าโปรแกรมหยุดการทำงานภายในฟังก์ชัน **findstop** ซึ่งถูกเรียกใช้โดยฟังก์ชัน main ( )

```
(dbx) stop in findstop
(2) stop in findstop
(dbx) run < testtabs
Running: tabs < testtabs
Stopped in findstop at line 38 in file
"/home/pairoj/cprogs/tabs.c"
        38 return (TABSIZE - (col % TABSIZE));
(dbx) print posn
posn = 3
(dbx) where
findstop(col=3), line 38 in "/home/pairoj/cprogs/tabs.c"
main(), line 17 in "home/pairoj/cprogs/tabs.c"
```

หลังจากทำการตรวจสอบข้อมูลที่ต้องการเป็นที่เรียบร้อย และต้องการให้โปรแกรมดำเนินการทำงานต่อไป ผู้เขียนโปรแกรมก็จะเรียกใช้คำสั่ง **cont** ซึ่งหลังจากโปรแกรมอ่านค่าข้อมูลจากไฟล์ **testtab** และแสดงผลให้ปรากฏบนหน้าจอแล้ว โปรแกรม **dbx** ก็จะแจ้งรายงานสิ้นสุดการทำงานของโปรแกรมและแสดงค่าคืนกลับที่ได้รับจากโปรแกรม **tabs** หลังจากเสร็จสิ้นกระบวนการดีบักและผู้เขียนโปรแกรมต้องการออกจากโปรแกรม **dbx** ก็สามารถทำได้โดยพิมพ์คำสั่ง **quit** ดังตัวอย่างต่อไปนี้

```
(dbx) cont
xyz     abc
execution completed, exit code is 33627
program exited with 91
(dbx) quit
```

เนื่องจากคำสั่งควบคุมการทำงานภายในโปรแกรม **dbx** มีอยู่ค่อนข้างมากและแยกออกเป็นหมวดหมู่หลายชุด หากผู้ใช้งานต้องการศึกษาความหมายของคำสั่งต่าง ๆ ก็สามารถทำได้โดยพิมพ์คำสั่ง **help** ขณะที่อยู่ในพรอมต์ (dbx) เพื่อแสดงรายการชุดคำสั่งซึ่งจำแนกออกตามประโยชน์การใช้งาน หากต้องการทราบรายละเอียดของคำสั่งใดก็สามารถพิมพ์คำสั่ง **help** ตามด้วยชื่อของคำสั่งที่ต้องการทราบนั้น

```
$ dbx
(dbx) help
    command summary
Execution and Tracing
    catch   clear   cont    delete  ignore  next    rerun
    run     status  step    stop    trace   when

.
.
The command 'help <cmdname>'
provides additional help for each command
(dbx)
```

สำหรับบทความในตอนต่อไปซึ่งเป็นตอนสุดท้าย ผู้เขียนจะกล่าวถึงรายละเอียดของ System Call และการใช้งานคำสั่ง make ปิดท้ายด้วยเรื่องของการบริหารซอร์สโค้ด ซึ่งหลังจากจบรายละเอียดทั้งหมดนี้แล้ว เชื่อว่าผู้ที่ติดตามบทความชุดนี้มาตั้งแต่ต้นคงจะมีความเข้าใจการเขียนโปรแกรมบนระบบปฏิบัติการ UNIX มากขึ้น สำหรับข้อคิดเห็นและข้อเสนอแนะต่าง ๆ กรุณาส่งมาที่ wpairoj@hotmail.com

พบกันฉบับหน้าครับ...

■ กู้เกียรติ ซื่อตรง

SOFT MAIL

การเขียนโปรแกรมบนวินโดวส์ด้วย MFC

: การจัดการกับแมสเสจและการพอร์ตโปรแกรมจาก Windows API

การเพิ่มตัวจัดการกับแมสเสจ รวมถึงการพอร์ตโปรแกรมจาก Windows API เพื่อมาใช้กับ MFC

ผมมักจะใช้เวลาหลังเลิกงานไปเดินตามร้านหนังสือใหญ่ ๆ ในกรุงเทพฯ เพราะร้านต่าง ๆ มักจะมีหนังสือดี ๆ ให้เลือกอ่าน (ฟรี) อยู่มากมาย จะไม่ให้อ่าน (ฟรี) ได้อย่างไรครับ หนังสือแต่ละเล่มนั้น ราคาเป็นเลข 4 หลักทั้งนั้น ถ้าเล่มไหนอ่านติดใจแล้วต้องการซื้อ ก็ต้องคิดหนักเหมือนกัน ตัวผมนั้นไม่เท่าไหร่หรอก แต่ผมเป็นห่วงเด็ก ๆ วัยรุ่นที่ชอบศึกษาหาความรู้ หาเทคนิคใหม่ ๆ คงจะเจ็บตัวน่าดูเหมือนกัน

คงจะโทษร้านค้าไม่ได้เช่นกัน เพราะราคาตามหน้าปกก็ประมาณ 40 ดอลล่าร์ คิดเป็นเงินไทยก็พันกว่าบาท ร้านหนังสือก็ขายตามราคานั้น ไม่ได้บวกเพิ่มค่าขนส่ง ค่าติดต่อกับสำนักพิมพ์ในต่างประเทศ ค่าอะไร ๆ อีกหลายอย่าง แถมบางร้านยังมีโปรโมชันให้สมัครสมาชิกฟรี และลดได้อีก 5-10 เปอร์เซ็นต์อีกด้วย

เรื่องของซอฟต์แวร์ก็เช่นเดียวกัน ผมเคยสนใจคอมไพเลอร์ของบริษัทหนึ่งเพราะเคยลองใช้รุ่นทดลอง (Trial Version) แล้วรู้สึกว่าดีมาก จึงไปค้นหารายละเอียดจากเว็บไซท์ของบริษัทผู้ผลิตและทราบราคารุ่นมาตรฐานประมาณ 80 ดอลล่าร์ จากนั้นได้ติดต่อไปที่ตัวแทนจำหน่ายในเมืองไทย (เป็นบริษัทที่มีชื่อเสียงมากในเมืองไทย) เขาขายในราคา 8,000 บาท ผมเลยไม่คิดที่จะซื้อ แต่ยังใช้คอมไพเลอร์ตัวนั้น ซึ่งยังเป็นรุ่นทดลอง ก่อนใช้ก็ต้องเปลี่ยนปีเป็นปี 97 เสียก่อนจึงจะใช้ได้ และไม่สามารถรันโปรแกรมที่เราสร้างโดยไม่รันคอมไพเลอร์ก่อน แต่ไม่เป็นไรครับ เพราะราคาเพียงร้อยกว่าบาท และก็ไม่ต่างอะไรมากนักกับรุ่นมาตรฐาน เรื่องราคาของซอฟต์แวร์เป็นเหตุผลสำคัญที่ทำให้ "ของโจร" ยังขายดีอยู่